A column generation approach to scheduling of parallel identical machines

Author: JULIA JOBSON
Supervisor: ELINA RÖNNBERG
Department of Mathematics, Linköping University
Examiner: OLEG BURDAKOV
Department of Mathematics, Linköping University

Date: 2019-08-29
Abstract

This thesis aims to implement a combination of Linear Programming Column Generation and a Large Neighbourhood Search heuristic to solve scheduling problems. The resulting method is named Integer Programming Column Search (IPCS). For computational evaluation, the IPCS method is applied to the problem Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources generalised to parallel identical machines.

The interest of combining exact procedures with heuristic approaches is quickly growing since scheduling problems have many and complex real-world applications. Most of these problems are NP-hard and therefore very challenging to solve. By using a combination of heuristic strategies and exact procedures, it can be possible to find high-quality solutions to such problems within an acceptable time horizon.

The IPCS method uses a greedy integer programming column generating problem introduced in a previous work. This problem is designed to generate columns that are useful in a near-optimal integer solutions. A difference to previously introduced method is that we here build a master problem, an Integer Programming Column Search Master (IPCS-Master). This is used to update the dual solution that is provided to the greedy integer programming column generating problem.

The computational performance of the IPCS method is evaluated on instances with 60, 70, 80, 90 and 100 jobs. The result shows that the combined design encourage the generation of columns that benefit the search of near-optimal integer solutions. The introduction of an IPCS-Master, which is used to update the dual variable values, generally leads to fewer pricing problem iterations than when no master problem is used.

**Keywords:** Scheduling, Parallel Identical Machines, Column Generation, Large Neighbourhood Search, Mixed Integer Programming, GCG
Acknowledgements

This paper is the result of my master thesis project and completes my education in Applied Physics and Electrical Engineering, Master of Science degree in Applied Mathematics at Linköping University.

The work has been carried out at the Department of Mathematics at Linköping University. I would like to thank my supervisor Elina Rönnberg for giving me the opportunity to work on such an interesting subject. I would like to thank you for your engagement and inspiration, for proposing the topic of the thesis and for being such a big part of carrying out this thesis work. It would not have been possible without your big-hearted effort.

I also would like to thank my examiner Oleg Burdakov for giving me feedback and support throughout this thesis work.

The computational experiments were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at National Supercomputer Centre (NSC).

Lastly, I would like to give sincere thanks for the motivation, inspiration, and engagement that the act of getting out in nature has given me when I have encountered difficulties and doubts.
Acronyms

API Application Programming Interface. 2, 42

CG Column Generation. 4, 8

D-W Dantzig-Wolfe. 11, 31

GIP-CG Greedy Integer Programming Column Generation. 8–10, 13, 14, 16, 22, 28, 40

IP Integer Programming. 5, 7, 19, 21, 22, 24, 27, 28, 33, 34, 41

IPCS Integer Programming Column Search. 1, 2, 13, 14, 16, 19–22, 24, 26, 28, 30, 31, 33–35, 38, 40–42

IPCS-Master Integer Programming Column Search Master. 13, 14, 16, 18, 19, 21, 22, 24, 27, 34, 36–38, 40–42

JSON JavaScript Object Notation. 16, 21

LNS Large Neighbourhood Search. 8, 9, 13, 14, 16, 17, 21, 41

LP Linear Programming. 4, 28

MIP Mixed Integer Programming. 2, 8, 11, 13, 21, 29, 31, 37

MP Master Problem. 6, 9, 29

PC-JSOCMSR Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources. 1, 4

RMP Restricted Master Problem. 6, 29

SUB Subproblem. 6, 8, 9, 29
# Contents

1 Introduction 1
   1.1 Purpose ......................................................... 1
   1.2 Aim .............................................................. 1
   1.3 Software ......................................................... 2

2 Theory 3
   2.1 Job Scheduling .................................................. 3
   2.2 Column Generation .............................................. 4
   2.3 Decomposition .................................................. 5
   2.4 Branch-and-Price .............................................. 7
   2.5 Large Neighbourhood Search .................................. 8
   2.6 Combination of Large Neighbourhood Search and Column Generation 8

3 Mathematical Models 10
   3.1 Column-oriented problem formulation .......................... 10
   3.2 Greedy Integer Programming Column Generation ............. 13
   3.3 Compact Formulation ............................................ 14

4 Implementation 16
   4.1 Procedure of The Combined Approach IPCS .................... 16
   4.2 The Dynamics of IPCS .......................................... 20
      4.2.1 Instances .................................................. 21
      4.2.2 Deriving the Strategy .................................... 21
      4.2.3 LP-CG Parameter Values .................................. 27
      4.2.4 LNS–CG Parameter Values ................................ 28
   4.3 LP Solution ....................................................... 28
      4.3.1 Standard Approach and Acceleration Strategies ........ 28
   4.4 GCG ............................................................. 31

5 Results 33
   5.1 Computational Results .......................................... 33
   5.2 Branch-and-Price .............................................. 37

6 Discussion 40
   6.1 Conclusion ..................................................... 40
      6.1.1 IPCS ....................................................... 40
      6.1.2 GCG ....................................................... 41
   6.2 Future Work .................................................... 41

References 43

Appendices 46

A Result tables 46
List of Figures

1  Scheduling example ........................................... 4
2  Procedure overview ........................................... 16
3  Program structure ............................................. 19
4  Average objective value of IPCS-Master using different $\gamma$-strategies  23
5  Distribution plots of IP objective value of IPCS-Master .............. 23
6  Recurrences in LNS-CG ......................................... 24
7  Distribution diagrams of regeneration of columns in LNS-CG .... 25
8  Distribution diagrams of regeneration of columns in IPCS-Master . 26
9  IP objective value for different runtime of the initiating LP-CG method 27
10 Absolute Gaps .................................................. 30
11 LP-CG method: LP-objective values and pessimistic absolute gap . 31
12 Result diagram ................................................... 34
13 Distribution of the solution quality ................................ 36
14 LNS best solution compared to IPCS-Master Solution .............. 37
15 Instances evaluated on IPCS and GCG ............................ 39

List of Tables

1  Instance number on which $\gamma$ strategies are evaluated on ......... 22
2  LP-CG termination criteria ...................................... 28
3  Instances used to evaluate IPCS .................................. 33
4  Results of Branch-and-price method ................................ 38
5  Results of IPCS method ........................................... 38
6  Result table for distribution diagrams .............................. 46

List of Algorithms

1  LNS ................................................................. 8
2  IPCS ............................................................... 20
1. Introduction

Today, scheduling can be found in all kinds of environments, especially where efficiency is high on the agenda. There are standard ways of solving these kinds of optimization problems. However, with the desire to efficiently solve large-scale problems, the interest in combining exact procedures with heuristic approaches is quickly growing. We can find application areas almost everywhere — such as scheduling of airplane crew, network communication and hospital machines, just to name a few. As Chen and Powell states in [1], scheduling problems have many and complex real-world applications. Unfortunately, most of these problems are NP-hard and therefore very challenging to solve. Using a combination of heuristic strategies and exact procedures makes it possible to find a solution to the problems within an acceptable time horizon.

1.1 Purpose

The purpose of this project is to investigate the potential in combining a mathematical programming technique for linear programs — Column Generation, and the heuristic approach — Large Neighbourhood Search. We name this combination Integer Programming Column Search (IPCS).

The class of problems that we use to evaluate this approach is scheduling problems, and more specifically Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources (PC-JSOCMSR) generalized to Parallel Identical Machines. This type of problem structure appears in the context of scheduling avionic systems (electronics within aircraft) [2].

IPCS is an extension of the method presented in [3]. There, the design of a mathematical model for combining Column Generation and a heuristic approach is derived. The article uses the heuristic Large Neighbourhood Search to evaluate the model. The pricing problem is designed to greedily generate columns that benefit the integer programming problem rather than the linear programming problem.

1.2 Aim

The aim is to implement a combination of Linear Programming Column Generation and a Large Neighbourhood Search heuristic. The contribution of this thesis is the implementation of IPCS and it includes the use of a Mixed Integer Programming solver and a heuristic search approach. A difference to [3] is that we here build a master problem (Integer Programming Column Search Master) which is used to update the dual solution that is provided to the greedy integer programming column generating pricing problem. The implemented method solves the problem of scheduling jobs on parallel identical machines. The design of IPCS and choice of parameter values are evaluated on instances of size of 60, 70, 80, 90 and 100 jobs. The implementation contains the following elements:
• A Linear Programming Column Generation algorithm
• A Column Generating Large Neighbourhood Search
• A construction of a restricted master problem using the columns generated by the column generating large neighbourhood search and the solution of its LP-relaxation to provide dual variable values to the pricing problem
• A derivation of a strategy for the dynamic parameter values in IPCS

For benchmarks:
• An implementation of an Integer Programming model used to calculate integer solution with columns generated by a Linear Programming Column Generation algorithm
• The use of the Branch-and-Price software GCG

The method and models presented and implemented during this work were proposed in a project proposal to the Swedish research council by Elina Rönnberg 2019. The details of the implementation and the computational results are a product of this thesis work.

1.3 Software

To solve Mixed Integer Programming (MIP) problems and Linear Programming problems we use Gurobi Optimizer and its MIP solver [4]. More specifically we use their Python object-oriented Application Programming Interface (API) for our implementation.

GCG is a comprehensive licensed open-source software that solves MIP problems with the Branch-cut-and-price method, released under the Lesser General Public License [5]. We intend to use this for benchmarking. GCG provides an interactive shell that solves MIP problems provided by the user with a one-line command [6].
2. Theory

This chapter introduces the concepts used to derive the algorithms presented in Chapter 3. Chapter 2.1 introduces the class of job scheduling problem that we address. Then, the three chapters 2.2, 2.3 and 2.4 introduce Column Generation. Chapter 2.5 presents the heuristic approach from [7] and Chapter 2.6 describes how Linear Programming Column Generation can be combined with the heuristic approach introduced in [3].

2.1 Job Scheduling

Job scheduling represents a strategy for finding an optimal sequence of jobs w.r.t. characteristics of the jobs and resource constraints. Such characteristics of jobs can be having a resource that are shared by all of the jobs and a secondary resource that are shared by only some of the jobs. Resource constraints can be for example that a job may only be processed on only one machine. There are three fields of characteristics which can be used to classify a job scheduling problem when the common resource is a machine, namely $\alpha|\beta|\gamma$, for details, see [8].

$\alpha$ - Describes the machine environment such as:

- single machine, only one machine to schedule
- identical parallel machines, there are $k$ identical machines where each job $j$ may be processed on any one of the $k$ machines
- uniform parallel machines, there are $k$ machines in parallel but the machines have different speed
- unrelated parallel machines, there are $k$ machines in parallel but each machine can process the jobs at a different speed.

$\beta$ - Describes specific characteristics of the jobs such as:

- deadline, each job $j$ included in a schedule must be completed before its deadline
- limited resources, each job $j$ require the use of one or multiple resources during the processing operation
- release dates, the earliest time at which job $j$ may start processing time, each job $j$ has a unit of time within which the process must be finished

$\gamma$ - Describes the optimality criterion of the problem

We consider the class of identical parallel machine scheduling with time windows, using the three fields notation we can refer to the problem as $P|r_j, \text{deadline}, M_j, s_{jk}|\sum \omega_j U_j$. $P$ denotes identical parallel machines, $r_j$ indicates the release date, $\text{deadline}$ implies a hard due-date constraint for each job, $M_j$ denotes the machine-eligibility restriction and $s_{jk}$ denotes the sequence-dependent setup time [9]. The objective is to maximize the overall profit, where $w_j$ is
assigned as the prize of each job [10] and the variable $U_j$ is 1 if job $j$ is processed within the constraints and restriction, 0 otherwise.

A visual example of the PC-JSOCMSR single machine problem, where resource 0 is shared by all of the jobs while resource 1, 2 and 3 are shared by only some of the jobs for their pre- and post-processing operations, can be found in Figure 1.

![Figure 1: Example of a schedule of jobs $S = \{1, 4, 7, 8, 10\}$ of a PC-JSOCMSR single machine schedule with $n = 10$ jobs to schedule and $m = 3$ secondary resources [2].](image)

### 2.2 Column Generation

We introduce a Linear Programming (LP) problem

$$\max \sum_{q \in Q} c^q \lambda^q \quad (1a)$$

$$\text{s.t. } \sum_{q \in Q} a^q \lambda^q \leq b \quad | \quad u \geq 0 \quad (1b)$$

$$\lambda^q \geq 0, \quad q \in Q \quad (1c)$$

and refer to it as the Master Problem. Assume that we have a feasible starting solution and a non-negative dual vector $u$ to this LP-problem. Performing an iteration of the simplex method for linear programs, one would look for an improving solution by examining the reduced costs $\bar{c}_q = c^q - u^T a^q$ of the non-basic variables to recognize if there is a basis better than the current. For large scale LP-problems, it may not be efficient to store all variables explicitly due to practical limits such as memory [11]. If $Q$ in the Master Problem is very large, that is the number of variables, then the Column Generation (CG) approach can be used. In a Restricted Master Problem
we start with a small subset \( Q' \subseteq Q \) of variables, also referred to as columns. Then, columns are generated as they can be used to improve the solution. An initial set of columns may not be at hand in the Restricted Master Problem at the starting point. In such case, a set of feasible solutions can be derived by using artificial variables or a heuristic approach. Following the steps of the CG algorithm, we would first obtain the primal and dual solution \( \lambda \) and \( u \), respectively, by solving the Restricted Master Problem. To proceed, a Subproblem is used to evaluate if there is an improving column \((c^q, a^q)\), \( q \in Q' \) from the non-empty set of feasible columns \( A \). Such column is found by solving the pricing problem

\[
\bar{c} := \max \{c(a) - u^T a \mid a \in A\},
\]

where \( c(a) \), the pricing function, represents the cost of the column: If the Subproblem no longer generates improving columns, that is, if \( \bar{c} \leq 0 \), the procedure stops and the current solution \( \lambda \) is an optimal solution to the Master Problem. However, as long as the Subproblem generates columns with \( \bar{c} > 0 \), the Restricted Master Problem is enlarged by adding the generated columns to the set \( Q' \) and the Restricted Master Problem is re-solved \([11]\). The step-wise procedure of CG is as follows:

1. (Initialization) Generate a starting set of columns for the Restricted Master Problem
2. Solve the Restricted Master Problem with the available columns. Let \( u \) be the values of the dual vector of the Restricted Master Problem.
3. Solve the Subproblem(s) using \( u \) from step 2.
4. If reduced cost \( \bar{c} \leq 0 \): stop iterating.
   If \( \bar{c} > 0 \): add the column \((c^q, a^q)\) to the Restricted Master Problem and return to step 2.

### 2.3 Decomposition

A special case of CG is Dantzig-Wolfe Decomposition, a method of special interest when the original problem is structured as a discrete problem. We let

\[
X = \{x \in \{0,1\}^n \mid Dx \leq d\}
\]

be a discrete bounded non-empty set and study the Integer Programming (IP) problem

\[
\max c^T x \quad \text{(3a)}
\]

\[
s.t. \ Ax \leq b \quad \text{(3b)}
\]

\[
x \in X. \quad \text{(3c)}
\]
The solution to (3) does not change if $X$ is replaced by its convex hull $\text{conv}(X)$. The replacement enables a representation of each $x \in X$ by a convex combination of extreme points of $\text{conv}(X)$. Let $x^1, \ldots, x^Q$ be the extreme points in $\text{conv}(X)$, then $x$ can be rewritten as:

$$x = \sum_{q=1}^{Q} \lambda^q x^q, \quad \sum_{q=1}^{Q} \lambda^q = 1, \quad \lambda^q \in [0, 1], \quad \forall q = 1, \ldots, Q$$

With the reformulation of $x$ and change of notations $c_q = c^T x^q, a_q = A x^q, q \in Q$, one may rewrite (3) and arrive to the formulation

$$\max \sum_{q \in Q} c_q \lambda^q \quad (4a)$$
$$\text{s.t. } \sum_{q \in Q} a_q \lambda^q \leq b \quad (4b)$$
$$\sum_{q \in Q} \lambda^q = 1 \quad (4c)$$
$$\sum_{q \in Q} \lambda^q x^q = x \quad (4d)$$
$$\lambda^q \geq 0, \quad \forall q \in Q \quad (4e)$$
$$x \in \{0, 1\}. \quad (4f)$$

An LP-relaxation of the Master Problem (MP) in (4) corresponds to relaxing the integrality of $x$ — relaxing $x$ gives no need for the coupling constraint (4d) linking $x$ and $\lambda$, and the constraint can be left out — and (5) is obtained.

$$\max \sum_{q \in Q} c^q \lambda^q \quad (5a)$$
$$\text{s.t. } \sum_{q \in Q} a^q \lambda^q \leq b \quad | \quad u \geq 0 \quad (5b)$$
$$\sum_{q \in Q} \lambda^q = 1 \quad | \quad w \quad (5c)$$
$$\lambda \geq 0 \quad (5d)$$

In the Restricted Master Problem (RMP) we have only some of the columns at hand, as in (2). RMP is solved to get a primal solution $\lambda$ and dual solutions $u$ and $w$, where $u$ corresponds to constraint (5b) and $w$ corresponds to the convexity constraint (5c). With values of these dual variables we solve the Subproblem (SUB), that is, the pricing problem:

$$\bar{c} := \max \{(c^T - u^T A)x - w \mid x \in X\}$$

A complete formulation of SUB is given in (6). Note that here, the integrality constraint (6c) of $x$ is still present.
\[ \max_{q \in \mathcal{Q}} \sum_{q \in \mathcal{Q}} \left( c^q - u^T a^q \right) x - w \quad (6a) \]
\[ \text{s.t. } Dx \leq d \quad (6b) \]
\[ x \in \{0, 1\} \quad (6c) \]

An optimal solution to SUB is an extreme point \( x^q \) of the convex hull of \( X \), and a solution can be added as a column \([c^q, a^q, 1]^T = [c^T x^q, (Ax^q)^T, 1]^T\) to RMP. The LP-relaxation of the original problem is solved when SUB no longer generates potentially improving columns \([11]\).

### 2.4 Branch-and-Price

When the CG method is applied to the LP-relaxation of an IP problem, it solves the problem over the convex hull of the feasible integer solutions, and the objective function is a real-valued function. If one is lucky, the problem has integrality property and the solution is an optimal solution to both the LP-relaxation and the IP problem, but that is generally not the case. If the LP-relaxation of an IP problem is solved by a CG method, the solution need not necessarily include a set of feasible columns for the IP problem. To generate an integer solution, one can combine CG with Branch-and-Bound and the strategy called Branch-and-Price is obtained \([11]\).

Branch-and-Bound is a tree-search method where a node represents a subproblem to the original problem. The subproblems are representing relaxations of parts of the original problem. For each node explored, one solves the subproblem and the objective value gives an optimistic bound for that part of the problem. The best feasible solution found during the search serves as a pessimistic bound for the complete problem. Land-Doig-Dakins is an LP-based Branch-and-Bound method that we here use as an example for explaining more about tree-search, for example branching and cutting decisions \([12]\). Using this method, one solves the LP-relaxation in each node and given the LP-solution, the branching is made on a variable with fractional value \( \bar{x}_j \). Branching on the fractional variable gives one branch for \( x_j \leq \lfloor \bar{x}_j \rfloor \) and another branch for \( x_j \geq \lfloor \bar{x}_j \rfloor + 1 \). The node is cut if a feasible solution is not found, if the LP-solution of the node is not better than best known pessimistic bound or if the solution is an integer solution. When there are no more branches to search, the algorithm stops. If an integer feasible solution is found during the search, then a solution to the IP problem is obtained.

Combining CG and Branch-and-Bound means that in each node, the LP-relaxation is solved by Column Generation. Branching on the fractional variable to include or exclude a column is difficult when using Branch-and-Price since the column can be regenerated by SUB. Therefore, branching strategies must be derived based on the underlying problem structure, examples of strategies for scheduling problems are found in \([1]\) and \([13]\).
2.5 Large Neighbourhood Search

Large Neighbourhood Search (LNS) is a meta-heuristic where the neighbourhood is the set of solutions that can be reached when first applying a destroy method and then repairing the destroyed solution. The destroy method destructs the current solution with some degree of destruction, which typically depends on the instance size, while the repair method rebuilds the destructed solution. The degree of destruction is an important design decision since a destroy method that destructs only a small part of the solution might not yield enough improvement and destroying too much of the solution might make the repair operation too challenging. The destroy method often involves a stochastic component that makes it possible to search the whole solution space. The destroy and repair strategies have some alternatives depending on the overall problem structure. If the repair method is to apply a MIP solver, then some parts of the solution are fixed and the rest are kept free so that the solution w.r.t. the free variables can be optimized [7].

For pseudo-code for a general approach of LNS, see Algorithm 1. There, $x^b$ is the best solution, which is initialized as the initial solution, and $x^c$ is the current solution, which is also initialized as the initial solution. The respective functions $\text{destroy()}$ and $\text{repair()}$ destructs and then repairs the current solution $x^c$. If the accept criteria of the repaired solution is met, the current solution $x^c$ is set as the repaired solution and if the current solution $x^c$ is better then the best solution $x^b$, the best solution is set as the current solution $x^c$.

Algorithm 1: LNS

\begin{algorithm}
\begin{algorithmic}
\STATE \textbf{Data:} feasible solution
\STATE \textbf{Result:} best Integer Solution found
\STATE $x^b = x_{\text{init}}$
\STATE $x^c = x_{\text{init}}$
\WHILE{criteria is met}
\STATE $x^d = \text{destroy}(x^c)$
\STATE $x^r = \text{repair}(x^d)$
\IF{$x^r$ meets acceptation criteria}
\STATE $x^c = x^r$
\ENDIF
\IF{$x^c$ is better then $x^b$}
\STATE $x^b = x^c$
\ENDIF
\ENDWHILE
\end{algorithmic}
\end{algorithm}

2.6 Combination of Large Neighbourhood Search and Column Generation

The mathematical formulation for the combination of LNS and CG are taken from [3]. The contribution of [3] is a design of an altered objective of SUB that generates columns useful to find a feasible and near-optimal integer solution. Using such SUB is called Greedy Integer Programming Column Generation.
The design of the SUB is such that it greedily finds the best column to potentially improve a current solution to MP. The Column-oriented LNS solution approach presented in [3] is one possible way of implementing the use of the GIP-CG and it is used in [3] to evaluate the performance of GIP-CG. The destroy method as described in [3] is a stochastic function that removes a column from an active set of columns in a solution to MP and the repair method uses the GIP-CG to generate a new column. The SUB algorithm is called \textit{Greedy Integer Programming Column Generation} since it uses a dual variable generated by, for example, linear programming column generation or subgradient optimization, and a penalty function to penalize the violation of some constraints. Using this GIP-CG design makes it possible to restrict the set of columns considered when solving the pricing problem. An application of the GIP-CG to parallel identical machines is introduced in Chapter 3.2.
3. Mathematical Models

This chapter introduces the mathematical models that are used in the implemented methods. The models in Chapter 3.1 are mostly taken from the literature and when modifications are made compared to the literature, this is specified. In Chapter 3.2, we use the design discussed in Chapter 2.6 when deriving the objective function to be used in GIP-CG.

3.1 Column-oriented problem formulation

How to make a column-oriented formulation for the problem to schedule jobs on parallel identical machines is taken from [13]. This article provides the decomposition that MP assign schedules to machines and SUB finds the best sequence of jobs w.r.t. constraints. We define a machine schedule as a sequence of jobs that together can be assigned to a single machine. A schedule is feasible if job $j$ is performed at most on one machine [13]. The mathematical model of MP is given in (7) with variables and parameters defined as follows.

**Notations:**
- $Q =$ the set of indices of the feasible schedules
- $k =$ number of machines available

**Parameters:**
- $t^q_j = 1$ if job $j \in J$ is in schedule $q \in Q$, 0 otherwise
- $c^q =$ profit of schedule $q \in Q$

**Decision variables:**
- $\lambda^q = 1$ if schedule $q \in Q$ is used, 0 otherwise

\[
\begin{align*}
\text{max} & \quad \sum_{q \in Q} c^q \lambda^q \\
\text{s.t.} & \quad \sum_{q \in Q} t^q_j \lambda^q \leq 1 \quad | u_j \geq 0 \quad j \in J \\
& \quad \sum_{q \in Q} \lambda^q \leq k \quad | w \geq 0 \\
& \quad \lambda^q \in \{0, 1\}
\end{align*}
\] (7a) (7b) (7c) (7d)
A schedule for one machine is a solution to the problem “Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources” introduced in [2]. This problem is formulated such that a common resource is shared by all jobs while some secondary resources are shared by some of the jobs. All jobs have a specific pre-processing time before using the common resource and a post-processing time after the use of the common resource. During pre- and post-processing times the job requires the secondary resource. Besides the resources, a job must be scheduled within one of its time windows, assuming that every job has at least one time window qualified to perform the whole process of the job. Each job has a prize and the problem is to find a subset of jobs to assign to a schedule to maximize the total prize of all jobs performed. A feasible schedule must take into account an overall time horizon. In the MIP-model proposed in [2], the use of both a common and some secondary resources is handled by sequence-dependent setup times, that is, a minimum time between the start of two jobs due to the sequencing. A modified version of the mathematical model introduced in [2] is presented in (8), where the pricing function is substituted with the reduced cost w.r.t constraints (7b) and (7c). The substitution of the objective function is problem specific and made to fit the Dantzig-Wolfe (D-W) decomposition. Parameters and variables of SUB are defined as follows.
Parameters:

- \( p_j > 0 \) is the processing time of job \( j \in J \)
- \( W_j = \bigcup_{\omega=1,...,\omega_j}[W_{j_{\omega}}^{start}, W_{j_{\omega}}^{end}] \) are the time windows of job \( j \in J \), where \( W_{j_{\omega}}^{end} - W_{j_{\omega}}^{start} \geq p_j, \omega = 1, \ldots, \omega_j, j \in J \)
- \( T_j^{rel} = \min_{\omega=1,...,\omega_j} W_{j_{\omega}}^{start}, j \in J \)
- \( T_j^{dead} = \max_{\omega=1,...,\omega_j} W_{j_{\omega}}^{end}, j \in J \)
- \( \delta_{jj'} = \) sequence dependent setup time between job \( j \) and job \( j', j, j' \in J \)

Decision variables:

- \( t_j = 1 \) if job \( j \) is in schedule, 0 otherwise, \( j \in J \)
- \( t_{j_{\omega}} = 1 \) if job \( j \) is assigned to time window \( \omega \), 0 otherwise, \( \omega = 1, \ldots, \omega_j, j \in J \)
- \( s_j = \) starting time of job \( j \)
- \( y_{jj'} = 1 \) if job \( j \) is scheduled before \( j' \), 0 otherwise, \( j, j' \in J, j \neq j' \)

Mathematical model for the pricing problem:

Based on the parameters and decision variables above the MIP formulation of the \( P|r_j, \text{deadline}, M_j, s_{jk}|\sum \omega_j U_j \) problem with one machine and modified objective function is given as:

\[
\text{max } \sum_{j \in J} (z_j - u_j)t_j - w \tag{8a}
\]

\[
\text{s.t } t_j = \sum_{\omega=1,...,\omega_j} t_{j_{\omega}} j \in J \tag{8b}
\]

\[
y_{jj'} + y_{j'j} \geq t_j + t_{j'} - 1 \quad j, j' \in J, j \neq j' \tag{8c}
\]

\[
s_j \geq s_j + \delta_j j' - (T_j^{dead} - p_j - T_j^{rel} + \delta_j j')(1 - y_{jj'}) \quad j, j' \in J, j \neq j' \tag{8d}
\]

\[
s_j \geq T_j^{rel} + \sum_{\omega=1,...,\omega_j} (W_{j_{\omega}}^{start} - T_j^{rel})t_{j_{\omega}} j \in J \tag{8e}
\]

\[
s_j \leq T_j^{dead} - p_j + \sum_{\omega=1,...,\omega_j} (W_{j_{\omega}}^{end} - T_j^{dead})t_{j_{\omega}} j \in J \tag{8f}
\]

\[
t_j \in \{0,1\} \quad j \in J \tag{8g}
\]

\[
t_{j_{\omega}} \in \{0,1\} \quad \omega = 1, \ldots, \omega_j \quad j \in J \tag{8h}
\]

\[
T_j^{rel} \leq s_j \leq T_j^{dead} - p_j \quad j \in J \tag{8i}
\]

\[
y_{jj'} \in \{0,1\} \quad j, j' \in J, j \neq j' \tag{8j}
\]
3.2 Greedy Integer Programming Column Generation

Here we give the mathematical model used in GIP-CG. As described in Chapter 2.2, a linear programming column generation method consists of a restricted master problem and a pricing problem. Solutions to the pricing problem correspond to variables – or columns – in the restricted master problem. By generating new columns, the solution space of the restricted master problem is enlarged. The pricing problem is provided with a dual solution obtained by solving the restricted master problem. The value of the dual solution has an influence on the objective function of the pricing problem and thereby an influence on which new columns that are generated.

In IPCS, the Integer Programming Column Search Master (IPCS-Master) is our restricted master problem and it is composed as (7). We use an LNS to improve a Current Solution by generating new columns using the GIP-CG pricing problem from [3]. All new columns are added to the IPCS-Master. By solving the LP-relaxation of IPCS-Master, the dual solution can be updated as in the linear programming column generation method. The value of the dual solution influences the search of finding the best column that potentially improve Current Solution. This work contribute with a design for combining the use of LNS and the construction of an IPCS-Master.

We know from Chapter 2.5 that the LNS heuristic uses destroy and repair methods to reach different parts of the solution space. From previous work we have that the destroy method can keep some parts of the solution, while the rest of the solution is destroyed. In our case, some parts of the Current Solution are destroyed by removing a prespecified number of variables, i.e machine schedules, while the rest of the variables are kept. Current Solution is then repaired by generating a solution to GIP-CG, i.e generating a new schedule which is obtained by solving GIP-CG with a MIP solver.

In IPCS method, a Current Solution only contains the active set of variables, corresponding to the non-zero variables in (7). The destroy method randomly selects a prespecified number of variables from the Current Solution to destroy. The rest of the variables

$$\hat{\lambda} \in \Lambda = \{\lambda^q \in \{0, 1\}, q \in Q : \sum_{\lambda \in Q} \lambda^q < m\}$$

(9)

in Current Solution are kept. A set

$$J_0 = \{j \in J : \sum_{q \in Q} t^q_j \hat{\lambda}^q = 1\} \subseteq J$$

(10)

includes the jobs that are included in the kept parts of Current Solution. Hence, the addition of a column that includes a job in $J_0$ would violate Constraint (7b). With this subset, we can create a penalty function

$$-M \sum_{j \in J_0} t_j,$$

(11)
that penalizes columns violating the constraint. Varying the value of the penalty parameter \( M \) intensify or diversify the search with respect to restricting the columns considered during the repair phase. Given a dual solution \( \tilde{u} \) to the LP-relaxation of IPCS-Master, we can create a GIP-CG type of pricing problem, which is used in the repair method. The cost function

\[
\sum_{j \in J} (z_j - \gamma \tilde{u}_j) t_j, \tag{12}
\]

together with the penalty function (11) and Constraints (8b)–(8j) define the pricing problem in our LNS. The cost used in (12) is a modification of the reduced cost with parameter \( \gamma \), such that \( \gamma = 1 \) makes the cost the same as reduced cost and \( \gamma = 0 \) makes the cost the same as the original. Therefore, we let \( \gamma \in [0, 1] \) such that the function is within the range of the original cost and the reduced cost. Letting \( \gamma \) be dynamic during the search has the consequence that columns can be regenerated. We therefore need to choose the values of \( \gamma \) carefully to avoid unnecessary regeneration of columns.

The pricing problem can use any dual solution, in [3] it is stated that the dual solution preferably is of high quality with respect to the LP-relaxation of the problem. How good the dual solution should be is one of the strategies that is to be derived. Strategies for our LNS and initial LP method are discussed in Chapter 4.2.

**Mathematical model of GIP-CG**

Following the design introduced in [3] and by combining the functions introduced above with Constraints (8b)–(8j), we obtain

\[
\max \sum_{j \in J} (z_j - \gamma \tilde{u}_j) t_j - M \sum_{j \in J_0} t_j \tag{13}
\]

s.t. Equations (8b)–(8j) are satisfied,

which defines GIP-CG used in the IPCS method.

### 3.3 Compact Formulation

A compact formulation of the problem capture the problem

\( P|r_j, \text{deadline}, M_j, s_{jk}| \sum \omega_j U_j \)

in one mathematical model. The compact formulation is needed as an input when computing a solution with the Branch-and-Price software GCG [6]. Decision variables and parameter values are the same as in the pricing problem Chapter 3.1, but with an additional index for each machine \( i \in I \). Putting an index on each machines causes symmetry within the model. Models containing symmetries are usually harder to solve. We use this model despite symmetry since we need a compact formulation for the GCG software and it is not in our focus to find a better compact model.
The mathematical model of the $P|\rho_j, \text{deadline}, M_j, s_{jk}|\sum \omega_j U_j$ problem is given as follows:

$$\text{max} \ \sum_{i \in I} \sum_{j \in J} z_j t^i_j$$

s.t. $$\sum_{i \in I} t^i_j \leq 1 \quad j \in J$$

$$t^i_j = \sum_{\omega = 1}^{\omega_j} t^i_{j,\omega} \quad j \in J, \ i \in I$$

$$y^i_{jj'} + y^i_{j',j} \geq t^i_j + t^i_{j'} - 1 \quad j, j' \in J, \ j \neq j', \ i \in I$$

$$s^i_{jj'} \geq s^i_j + \delta_j j' - (T^\text{rel}_{jj'} - p_j - T^\text{rel}_{j,j'} + \delta_j j')(1 - y^i_{jj'}) \quad j, j' \in J, \ j \neq j', \ i \in I$$

$$s^i_j \geq T^\text{rel}_{jj} + \sum_{\omega = 1}^{\omega_j} (W^\text{start}_{jj\omega} - T^\text{rel}_{jj\omega}) t^i_{j,\omega} \quad j \in J, \ i \in I$$

$$s^i_j \leq T^\text{dead}_{jj} - p_j + \sum_{\omega = 1}^{\omega_j} (W^\text{end}_{jj\omega} - T^\text{dead}_{jj\omega}) t^i_{j,\omega} \quad j \in J, \ i \in I$$

$$t^i_j \in \{0, 1\} \quad j \in J, \ i \in I$$

$$t^i_{j,\omega} \in \{0, 1\} \quad \omega = 1, \ldots, \omega_j, \ j \in J, \ i \in I$$

$$T^\text{rel}_{jj} \leq s^i_j \leq T^\text{dead}_{jj} - p_j \quad j \in J, \ i \in I$$

$$y^i_{jj'} \in \{0, 1\} \quad j, j' \in J, \ j \neq j', \ i \in I$$
4. Implementation

The implementation chapter goes through the details of the design for the implementation of the IPCS method. The mathematical models used in this method are described in Chapter 3. We make a step by step description of the strategies for the dynamic parameter values derived. In Chapter 4.3, we explain how we derived an LP-solution that we use to evaluate the result of IPCS. Then, the use of the GCG software is presented in Chapter 4.4.

Figure 2 gives an overview of the heuristic method IPCS. LP-CG refers to the standard linear programming column generation method from Chapter 2.2. There, the jobs are included in a schedule by a pricing problem, SUB, and then the schedules are assigned to a machine in a restricted LP-Master as explained in Chapter 3.1. LP-CG generates columns that initialize the IPCS-Master and our Current Solution in LNS-CG. The dual solution of LP-Master initialize our pricing problem GIP-CG that we use in LNS-CG. The LNS-CG method is built on an LNS that improve our Current Solution by solving the pricing problem to generate new potentially improving columns. The generated columns are also added to IPCS-Master. By solving its LP-relaxation, new dual values are obtained and are used to update the objective in GIP-CG. The final result is an integer solution found by solving the IPCS-Master with all the generated columns from both LNS-CG and LP-CG. During the search, the collected data is saved to a JavaScript Object Notation (JSON) database.

![Figure 2: Simplified overview of IPCS](image)

4.1 Procedure of The Combined Approach IPCS

IPCS can be divided into 3 different parts which transfer information between each other. The different methods are separately described and then a full
overview of the information flow between the methods is presented in Figure 3.

LP-CG

LP-CG is an LP Column Generation method organized as in the following steps:

1. Generate an initial LP-Master
   The initializing method generates the Restricted LP-Master by solving a single machine problem with the original cost in the pricing function. This provides the Restricted LP-Master with one initiating column. It may be more efficient to initialize with more than one column as in [13], but this is not studied in this thesis.

2. Solve the LP-relaxation of the Restricted LP-Master
   The relaxation of the Restricted LP-Master, implemented as in (7), is solved with Gurobi Optimizer and the dual solution is handed to the pricing problem — SUB.

3. Generate a column
   The objective function of SUB is updated with the dual variable values from the Restricted LP-Master. By solving SUB, implemented as in (8), with Gurobi Optimizer, one obtains a schedule that is added as a column to the Restricted LP-Master.

4. Termination of the LP-CG method
   LP-CG iterates between step 2 and step 3 until a good enough dual solution to the LP-relaxation of the Restricted LP-Master is found.

Since SUB can be time-consuming to solve, both a time and MIP-Gap limit is implemented — The MIP-Gap is the relative difference between the optimistic and pessimistic bound during Branch-and-Bound in Gurobi Optimizer. This means that SUB usually terminates before an optimal solution to the single machine problem is found. Therefore, the reduced cost is not maximized in each iteration as it would in a standard linear programming column generation procedure. Hence, one can not use the common solution bound to evaluate the quality of the primal and dual solution. This is further described in Chapter 4.3.

LNS-CG

LNS-CG is a Column Generating Large Neighbourhood Search method introduced in [3]. We use a LNS to generate new columns with the pricing problem (13) to improve a Current Solution. Our LNS improves the Current Solution performing a destroy method, the stochastic component, implemented such that one column is removed from Current Solution by deleting a variable. A repair method, repair the destroyed Current Solution by generating a new column solving the pricing problem with Gurobi Optimizer. If the new column violates any of the constraints in the destroyed Current Solution — the pricing function generates a new column for the same deleted variable until all constraints are satisfied. For each call of the repair method, the $\gamma$- and $M$-values are updated. Termination of LNS-CG
depends on the value of $\gamma$. All columns generated by the pricing problem are collected and saved in a column collector until the search has reached termination. The following steps organize the method:

1. Initialize LNS-CG
   Current Solution in LNS-CG is initialized with columns generated by LP-CG. The initializer inserts the columns in the order which they have been generated. If a column violates any constraints, the next column is checked. If all following columns violate some constraints, columns with constants equal to zero are added until there are as many columns as machines available. Also, the Current and Best Solution is set to the initial solution. The initializing step is only performed in the first call of LNS-CG.

2. Destroy Current Solution
   The destroy function picks a random number between zero and the number of machines minus one to define which variable of Current Solution to destroy. The variable is then deleted from Current Solution.

3. Repair the destroyed Current Solution
   The repair function generates a new column for the destroyed Current Solution by solving the pricing problem implemented as (13). Before generating a new column, $\gamma$ and $M$ are updated according to the derived strategy in Chapter 4.2. Jobs still scheduled in the destroyed Current Solution are added to the set $J_0$. Then, the objective function of the pricing problem is updated with new parameter values and solved with Gurobi Optimizer.

4. Add variable
   If the generated column violates any constraints and is not already in the column collector, save the column in column collector and go back to step 3. If all constraints are satisfied add the variable to Current Solution and save the column in column collector.

5. Set Best Solution
   If the Current Solution is better then the Best Solution, reset the Best Solution to the Current Solution.

6. If termination criteria are not satisfied go back to step 2, otherwise go to next step.

7. Pass the generated column collector saved in LNS-CG method to IPCS-Master and terminate LNS-CG.

IPCS-Master

IPCS-Master is the extension of the implementation that collects columns generated by both LP-CG and LNS-CG. With the collected columns, IPCS-Master updates the dual variable values of the pricing problem in LNS-CG by solving the LP-relaxation. The pricing problem in LNS-CG uses the updated dual solution to repair the Current Solution. The procedure is organized as follows:
1. LP-CG
Run the LP-CG method to retrieve columns and good enough dual values. The retrieved columns are added to IPCS-Master, implemented as in (7).

2. Initialize LNS-CG
IPCS-Master initialize LNS-CG by adding columns, generated by LP-CG, to the Current Solution and initialize the pricing problem with the dual variable values.

3. LNS-CG
Run LNS-CG method until termination. When the method terminates the columns saved during LNS-CG are added to IPCS-Master. A column from the column collector is only added to IPCS-Master if it is not identical to any of the existing columns.

4. Solve LP-relaxation of IPCS-Master
The IPCS-Master and its LP-relaxation with all columns generated (including those from LP-CG) are solved with Gurobi Optimizer to update the dual values and obtain an IP solution to the problem.

5. Update the pricing problem in LNS-CG
Update the pricing problem in LNS-CG with the new dual values.

6. If the maximal number of LNS-CG iterations have been performed, terminate IPCS.
   If the maximal number of iterations has not been reached, go back to step 3.

Figure 3 gives an overview of the information transferred between the methods.
Algorithm 2 is an overview of the complete IPCS method.

<table>
<thead>
<tr>
<th>Algorithm 2: IPCS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data:</strong> Number of machines available, Instance data for single machine problem</td>
</tr>
<tr>
<td><strong>Result:</strong> Best integer solution found</td>
</tr>
<tr>
<td><strong>LP-CG</strong></td>
</tr>
<tr>
<td>while termination time is not met do</td>
</tr>
<tr>
<td>Solve the LP-relaxation of RMP</td>
</tr>
<tr>
<td>Update objective function of SUB</td>
</tr>
<tr>
<td>Solve SUB</td>
</tr>
<tr>
<td>if reduced cost $\leq 0$ then</td>
</tr>
<tr>
<td>terminate</td>
</tr>
<tr>
<td>Update RMP with new column</td>
</tr>
<tr>
<td>Initialize IPCS-Master and Current Solution with LP-Master columns</td>
</tr>
<tr>
<td>Initialize GIP-CG with LP-Master dual solution</td>
</tr>
<tr>
<td>while maximum iterations is not met do</td>
</tr>
<tr>
<td>LNS-CG</td>
</tr>
<tr>
<td>while $\gamma &gt; 0.2$ do</td>
</tr>
<tr>
<td>if Number of variables in Current Solution = Number of machines available then</td>
</tr>
<tr>
<td>Destroy Current Solution</td>
</tr>
<tr>
<td>Update $M$ and $\gamma$ values</td>
</tr>
<tr>
<td>Solve GIP-CG</td>
</tr>
<tr>
<td>if Column meet constraints of Current Solution then</td>
</tr>
<tr>
<td>Add new column to Current Solution</td>
</tr>
<tr>
<td>Save new column to column collector</td>
</tr>
<tr>
<td>if Current Solution better than Best Solution then</td>
</tr>
<tr>
<td>Set Best Solution to Current Solution</td>
</tr>
<tr>
<td>Add new columns to IPCS-Master</td>
</tr>
<tr>
<td>Solve IPCS-Master and its LP-relaxation</td>
</tr>
<tr>
<td>Update objective function of GIP-CG with new dual solution</td>
</tr>
</tbody>
</table>

4.2 The Dynamics of IPCS

Here we present how to choose the dynamic parameter values and the search strategies derived for the IPCS method. The choices we consider during the implementation are: when to terminate Gurobi Optimizer solving the pricing problems, if $\gamma$ and $M$ should be dynamic or static and what strategy to use for choosing those values. **How many columns** to be destroyed and repaired in each LNS-CG iteration. **The quality** of the initial dual variable values $\tilde{u}$. 

20
4.2.1 Instances

When deriving IPCS method it is evaluated on instances taken from [14]. Since the problem is solved for “identical parallel machines” we use instances made for $P|r_j, \text{deadline}, M_j, s_j|\sum \omega_j U_j$ single machine problems. For a detailed description of the instances, see [14].

When referring to the size $n$ of an instance we refer to the number of jobs that potentially can be scheduled, and the letter $m$ is associated to the number of secondary resources (an instance is formulated such that a common resource is shared by all jobs while some secondary resources are shared by some of the jobs). For example, an instance named $n0070\_m3\_000$ has 70 jobs, uses 3 secondary resources and the instance has the number 000. There are instances with 3, 4 and 5 secondary resources and for every number of secondary resources there are 30 instances numbered from 000 – 029 and we use instances of size 10 – 100 (incremental increase with ten jobs). So, in total there are 900 instances to use when deriving a strategy for IPCS.

4.2.2 Deriving the Strategy

To be able to understand how the design for the implementation of the IPCS method works, it is tested first on smaller instances and with time some of the larger instances. The number of ways to choose and combine different parameter values is huge. Therefore, the derived strategy is one of many possible ways to specify the parameter values. To evaluate the performance of the IPCS method, we save data to a JSON database and with the saved data we can visualize some of the important performance markers in diagrams. Such diagrams can, for example, visualize the number of column recurrences during LNS-CG, Best LNS solution found and how the dual values have changed.

**Gurobi Optimizer termination**

Terminating Gurobi Optimizer before an optimal solution is found when solving a MIP problem means that Gurobi Optimizer returns the best integer solution found during branch-and-bound. In our case, this is important since both the pricing problems in LP-CG and LNS-CG are very time consuming to solve to optimality and sometimes it is not possible to reach an optimal solution within a reasonable time horizon. In our case, we are interested in when the best integer solution found during branch-and-bound gives a positive cost to the pricing problem — where a positive cost is defined as a value greater than 0.01. By evaluating the IPCS method on different instances, we can see that if Gurobi Optimizer terminates when the first positive column is found, many columns get regenerated. Also, we see only small changes in dual values in-between IPCS-Master iterations. While having a time limit where Gurobi Optimizer runs at least 20 s gives a better IP solution and fewer re-generations of columns. If an optimal solution is found before 20 s, Gurobi Optimizer terminates automatically.
**Penalty parameter M**

Penalty parameter $M$ is a part of the penalty function (11) in the pricing problem used in LNS-CG. To dynamically change $M$ values can diversify and intensify the search with respect to the columns considered during the repair phase. To derive the strategy for $M$, we evaluate the IPCS method with constant small and big values of $M$ to see how the performance varies. As one may imagine, the greater value of $M$, the faster the pricing problem is to solve. The idea to change the value of $M$ with the number of iterations comes from [3]. Letting $M$ be dynamic and increase with the number of pricing problem iterations gave a good result, therefore the function

$$M = (\text{mean(original job prizes)} + 1) \times (0.5 \times \text{number of pricing problem iterations})$$

was derived to update $M$ in each iteration of GIP-CG in LNS-CG.

**$\gamma$ step-size**

$\gamma$ is the parameter that controls the influence of the dual variable values in the pricing problem in LNS-CG, constructed as (13). $\gamma = 1$ corresponds to the reduced cost while $\gamma = 0$ corresponds to the original one. During the initial test phase we let $\gamma$ decrease from 1 to 0 with a step-size of 1% for each iteration, to get an overall understanding of how the $\gamma$ value influences the performance of IPCS with respect to IP objective value, dual value changes and runtime. The data indicate that too small values of $\gamma$ does not improve IP objective values. Therefore, we let the $\gamma$ values stay within the interval $[0, 1]$. After initial tests and further understanding of how the $\gamma$ values influence the performance of IPCS, we let the step-sizes be constant and the tests are made with step-sizes 0.1, 0.08 and 0.06. The IPCS method with the different strategies are evaluated on instances in Table 1. Figure 4 illustrate the average IP objective value with respect to instance size in each IPCS-Master iteration. The distribution of the IP objective values for the different strategies of the instances in Figure 5 gives a clear illustration that step-size = 0.1 generally gives a slightly better IP objective value.

<table>
<thead>
<tr>
<th>Size</th>
<th>$m = 3$</th>
<th>$m = 4$</th>
<th>$m = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>70</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>80</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
</tbody>
</table>

Table 1: Instance number on which $\gamma$ strategies are evaluated on
(a) Size: 60 jobs, 2 machines
(b) Size: 70 jobs, 2 machines
(c) Size: 80 jobs, 2 machines
(d) Size: 80 jobs, 3 machines

Figure 4: Average objective value of IPCS-Master using different $\gamma$-strategies

(a) Distribution plot of IP objective
(b) Distribution plot with respect to value categories

Figure 5: Distribution plots of IP objective value of IPCS-Master
Evaluating the IPCS method with different $\gamma$ step-sizes gives a varying number of regeneration of columns. Figure 6 depicts an evaluation of $\gamma$ step-size by a number of average recurrences of columns within the same instance size during an iteration of LNS-CG. The number of recurrences are generally lower for $\gamma$ step-size $= 0.1$, which can be seen in Figure 7. The figure illustrates the regeneration within the same strategy. Since the performance of IPCS with respect to IP objective value showed small differences among the strategies, step-size $= 0.1$ is the choice of strategy. Since the column collector in LNS-CG is cleared for every IPCS-Master iteration it is also possible to regenerate columns for IPCS-Master. This recurrence of columns are depicted in Figure 8 where the differences within the different strategies are very small.

![Average recurrence of columns in LNS-CG](image1)

(a) Size: 60 jobs, 2 machines  
(b) Size: 70 jobs, 2 machines

![Average recurrence of columns in LNS-CG](image2)

(c) Size: 80 jobs, 2 machines  
(d) Size: 80 jobs, 3 machines

Figure 6: The figure depict how many times GIP-CG average regenerate columns during LNS-CG
(a) Distribution plot of number of regenerated columns in LNS-CG method

(b) Distribution of number of recurrences in LNS-CG method categorised according to strategy

Figure 7: Distribution diagrams of regeneration of columns in LNS-CG
(a) Distribution plot of number of regenerated columns in IPCS-Master

(b) Distribution of number of recurrences in IPCS-Master method categorised according to strategy

Figure 8: Distribution diagrams of regeneration of columns in IPCS-Master

\(\bar{\lambda}\) Removal
Since the instances considered in this thesis are evaluated on 2 or 3 machines, the decision to only remove one variable from Current Solution is made without testing other possibilities. If the pricing problem in LP-CG were easier to solve, or a heuristic procedure was implemented to get a good solution in a shorter time, larger instances with a higher number of machines available could have been evaluated. Then, investigating the performance of removing more than one variable from Current Solution is of great interest.

\(\hat{u}\) value
Different quality of the initial \(\hat{u}\) is tested by evaluating the IPCS method when varying the termination criteria of the initializing LP-CG method. The evaluation
of the termination is made such that we look at the IP objective value compared to the number of pricing problem iterations. Figure 9 depicts the IP solution of IPCS-Master where the lines represent different runtime in seconds of the initializing LP-CG method. As illustrated in Figure 9, the longer LP-CG runs the more pricing problem iterations are needed to reach higher IP objective values. The result is interpreted such that with the introduction of the extension IPCS-Master, which updates the dual variables, the need of an initial dual solution of high quality is no longer necessary. Since this is the case, we let the pricing problem in LP-CG run for a lower number of iterations.

4.2.3 LP-CG Parameter Values

During the LP-CG method, there are a few strategic decisions made. The initialization of LP-Master can be seen as an influencing decision, but this is not studied in this thesis. The first decision made is when to terminate the search of a new column, and this has to be made since the pricing problems are very time consuming to solve for larger instances. Gurobi Optimizer terminates when a positive, column cost $> 0.01$, is found and the search duration has reached at least 20 s. The second decision made is the termination of LP-CG method. When to terminate LP-CG method influence the quality of the dual variable values $\tilde{u}$. Table 2 lists the approximate number of seconds LP-CG runs before terminating.
Table 2: The table lists the runtime $t[s]$ of LP-CG before terminating and passing dual values to IPCS-Master and initializing LNS-CG

<table>
<thead>
<tr>
<th>Instance size</th>
<th>nr. jobs</th>
<th>LP–CG Termination $t[s]$</th>
<th>nr. Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>200</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>300</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>300</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>400</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>400</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>600</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>600</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>800</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>800</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

4.2.4 LNS–CG Parameter Values

Going through LNS-CG procedure we start by destroying Current Solution by removing one variable. After removal, the objective function of GIP-CG is updated with a new penalty value depending on what jobs that are still in the destroyed Current Solution and the $\gamma$ value decreases. With the updated parameter values, the procedure repairs the solution by solving the pricing problem to generate a new column. The value of $\gamma$ is updated such that its value decrease from 1 with step-size 0.1. LNS-CG terminates when $\gamma < 0.2$. The value of penalty parameter $M$ increases according to function (15). Since the pricing problem varies in runtime depending on the penalty function we also set Gurobi Optimizer to terminate when a column with cost $> 0.01$ is found and the runtime is at least 20 s.

4.3 LP Solution

In this chapter we explain how to derive a strategy to compare the result of IPCS. A very good comparison would be to compare to a standard IP problem method, as described in Chapter 4.4. But we have, with the given resources, not found a way to solve the problems to optimality, neither the IP problem or its LP-relaxation. The reason for this is how computationally challenging the original single machine scheduling problem is to solve. This problem is extremely time consuming to solve, as mentioned in previous chapters. Instead this chapter explains how to derive a good LP solution with a standard method and derive a feasible IP solution – based on that.

4.3.1 Standard Approach and Acceleration Strategies

We compare the result of IPCS to a solution derived by solving an IP problem over the columns generated when solving the LP-relaxation by column generation method. However, the convergence of column generation is slow and may take a
very large number of iterations before LP-optimality is proven. Therefore, one can use an approximate solution that stops if the current LP-solution is within a prespecified percentage of optimality. This is possible to perform since in each iteration one can calculate an interval within which we know optimality can be found.

Let \( \bar{z} = u^T b + w \), where \( u \) and \( w \) are dual solution with respect to Constraint (7b) and (7c) respectively, be the objective function value of RMP. Then, \( z^*_\text{MP} \geq \bar{z} \), and \( \bar{z} \) is a lower bound on \( z^*_\text{MP} \). When an upper bound \( m \geq \sum_{q \in Q} \lambda^q \) in (7), holds for an optimal solution of MP, we can use this to calculate an upper bound in each iteration: \( \bar{z} \) cannot increase more then \( m \) times the maximum value of the reduced cost \( \tilde{c}^* = \max\{ (c^T - u^T A)x - w \mid x \in X \} \), in each iteration. Hence,

\[
\bar{z} \leq z^*_\text{MP} \leq \bar{z} + m\tilde{c}^*. \tag{16}
\]

However, we cannot compute \( \tilde{c}^* \) since SUB is too computational challenging. Instead of finding the most positive reduced cost we compute a column with any positive reduced cost \( \bar{c} \). Let \( \bar{c}_{\text{UB}} \) be the optimistic bound during branch-and-bound in Gurobi Optimizer solving SUB. Then, when terminating according to our criteria, we get an upper and a lower bound on the maximal reduced cost within which we know \( \tilde{c}^* \) can be found. This MIP-Gap between the pessimistic and optimistic bound is in our case typically huge.

Instead of having the desired absolute gap on MP in (16) we can, by using the optimistic bound on SUB from (17), obtain a pessimistic absolute gap on MP by constructing the interval

\[
\bar{z} \leq z^*_\text{MP} \leq \bar{z} + m\bar{c}_{\text{UB}} \tag{18}
\]

This pessimistic interval makes it possible to terminate the algorithm when it finds a solution within a prespecified percentage of optimality.

Since the MIP-Gap typically is large, and very hard to decrease because of the difficulty of the pricing problem, the pessimistic absolute gap can be large and sometimes to large to use as termination criteria.

There is also another termination criteria that is based on a phenomenon called the \textit{tailing-off effect}. That is, a slow convergence when the objective function value of RMP does not improve sufficiently in a given number of iterations. Figure 11a depict a pessimistic absolute gap using the interval derived in (18). The \textit{tailing-off effect} are very clearly depicted in Figure 11b. In the figure, the standard deviation of the last 100 LP-objective values equals 0.2848, which in our case is a sufficiently small change for such a large number of SUB iterations.

We can combine the termination within an absolute gap on MP with the search of hitting the \textit{tailing-off effect}. But, because of the very hardness of the problem all instances did not reach such low value of the standard deviation. So, we also
implement the option of terminating the column generation after a certain time limit. For instances with a size smaller than or equal to 90 we implement a time limit of 24 hours, while for instances of size 100 we implement a time limit of 48 hours. We use both the absolute gap and the standard deviation of the last 100 objective values when considering if the results are good enough to use. When a good enough LP-relaxation is obtained, an integer solution is found by solving the IP problem with the generated columns. The IP-solutions found were then compared to the IPCS results by calculating the percentage to the best LP-solution found.

Figure 10: This Figure depicts the relation between different types of absolute gaps
4.4 GCG

The GCG software is a Branch-and-Price software for MIP problems and is a part of the SCIP Optimization Suite framework. GCG perform a D-W decomposition on a given problem and solves it with the Branch-and-Price method. The given problem may be given together with a decomposition structure, and in the case that the decomposition structure is not given by the user the program can automatically detect a structure. GCG provides the possibility to use an interactive shell but also the possibility to program plug-in modules. The intention of using the software was to get a benchmark to IPCS. To use it as desired was harder than predicted and we have therefore only made some preliminary experiments. With more work, the GCG software would probably be of great use as a benchmark. But, with the shortage of time, we used the provided interactive shell by one-line command to the terminal, which performs the following:

1. Read compact instance file
2. Read decomposition file
3. Optimize
4. Terminate when time limit hits
5. Write solution and statistic to solution files
6. Quit

In the solving process, SCIP uses SoPlex 4.0.1 as the default LP-solver, which we rather would have changed to Gurobi Optimizer to give a more accurate comparison. As mentioned in Chapter 3.3, the model contains symmetries. GCG states on the website that “If the same variables appear in the same order with the same coefficients in the same constraints for all pricing problems (and with the same coefficients in the master constraints), GCG will detect that and aggregate the pricing problems” [6]. However, we have not found any indication of aggregation even if we believe that our pricing problems are identical. In interest
of time, it is left for future work to use GCG more efficiently in order to provide a better benchmark.
5. Results

The computational experiments are performed on Tetralith, the largest of the National Supercomputer Centre’s (NSC) High-Performance Computing (HPC) cluster. Tetralith is funded by Swedish National Infrastructure for Computing (SNIC), for research by Swedish research groups. Every computer node, on which the computational experiments are performed, have 32 CPU cores with 96 GiB RAM and a local SSD disk where applications can store temporary files. We use Gurobi Optimizer default settings which means that generally, all available CPU cores are used, but it may choose to use fewer. The methods are implemented in Python 3.

5.1 Computational Results

IPCS is evaluated on the instances listed in Table 3. These are randomly selected from https://www.ac.tuwien.ac.at/research/problem-instances/ by generating a number between 0 – 29 with `randrange` in Python library `random`. For comparison we also derive a near-optimal LP-solution for each instance in Table 3 using the linear programming column generation method LP-CG, as described in Chapter 4.3. With the generated columns we solve the IP problem with Gurobi Optimizer — this is referred to as the LP-CG and IP method. The result and comparison of the methods are illustrated in the figures in this chapter.

<table>
<thead>
<tr>
<th>Size</th>
<th>$m = 3$</th>
<th>$m = 4$</th>
<th>$m = 5$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>011</td>
<td>024</td>
<td>009</td>
<td>2</td>
</tr>
<tr>
<td>70</td>
<td>023, 011</td>
<td>010, 005</td>
<td>013, 020</td>
<td>2, 3</td>
</tr>
<tr>
<td>80</td>
<td>010, 001</td>
<td>022, 006</td>
<td>015, 010</td>
<td>2, 3</td>
</tr>
<tr>
<td>90</td>
<td>023, 0025</td>
<td>006, 023</td>
<td>019, 007</td>
<td>2, 3</td>
</tr>
<tr>
<td>100</td>
<td>020, 017</td>
<td>014, 020</td>
<td>015, 018</td>
<td>2, 3</td>
</tr>
</tbody>
</table>

Table 3: Instances used to evaluate IPCS

At 163 pricing problem iterations instance $n0070.m3.011$ reaches LP-optimality using LP-CG, hence the algorithm then terminates and do not generate more columns for the IP problem. We have therefore made the data for this instance constant for the remaining pricing problem iterations in the figures throughout this chapter.

We measure the quality of an instance’s integer solution by calculating the percentage of its LP-solution, as in (19).

$$\frac{\text{integer solution}_{\text{instance } x}}{\text{best LP-solution found}_{\text{instance } x}} \times 100 = \% \text{ percentage}_{\text{instance } x}.$$ (19)
Figure 12 compare the quality of the solutions to IPCS-Master and the IP problem in LP-CG and IP at every ninth pricing problem iteration. The horizontal axis represents the average quality of the solutions within the same instance size, independent of the number of machines available. The vertical axis represents the number of pricing problem iterations needed to reach such quality. The dotted lines are the result of the LP-CG and IP method, solid lines are the result of the IPCS method and the colors separate instance size. The solid lines do not start at 9 iterations because of the initial LP-CG method, see Chapter 4.1 for implementation details of the IPCS method. After the initial LP-CG method the IPCS algorithm perform an iteration of LNS-CG, hence the big gap between the quality of the solutions from the different methods in Figure 12.

There is a large difference in performance between the IPCS method and the LP-CG and IP method. As we can see in Figure 12, IPCS reaches a high-quality solution after about 100 pricing problem iterations, while the LP-CG and IP method gradually improve and do not necessary generate columns included in a high-quality integer solution.

Figure 12: The figure illustrate the results of IPCS and LP-CG and IP by drawing the average quality of the integer solutions within the same instance size.

Figure 13 illustrate two distributions made on the quality of the results after approximately 477 pricing problem iterations. The data used in the figure can be found in Table 6 in appendices. The approximation of pricing problem iterations
is needed because of the initial LP-CG method in IPCS, which adds between 10 – 40 iterations to the 477 iterations depending on instance size. The distributions are based on the solution quality categorized according to instance size, method and number of machines available. The comparison of the methods shows that the integer solution of the IPCS method is of high-quality independent of instance size, while the LP-CG and IP method give varying solution quality. When comparing the number of machines available we can see that both methods perform slightly better when there are 3 machines available compared to 2 machines. This is to be expected since the case of 3 machines give more possibilities to schedule jobs.
Figure 13: Illustrate the distribution of the solution quality after approximately 477 pricing problem iterations. The distributions are made with the default settings of the boxplot function in the seaborn library, categorized on method and number of machines available.

Figure 14 illustrate the difference in performance between IPCS-Master solution and the Best LNS Solution found. The average quality of IPCS-Master solutions are drawn with dotted lines and the average quality of the Best LNS Solutions are
drawn with solid lines. The colors separate instance sizes, independent of the number of machines available. We can see in the figure that by solving IPCS-Master we do not yield a solution of higher quality than what the Best LNS Solution finds provided that the dual solution is of high quality, which seems to be the case after updating the dual variables approximately 30 times. However, using IPCS-Master to update the dual solution is of importance as we could see in Figure 9, since it leads to fewer pricing problem iterations than if the LP-CG method were used to derive the quality of the dual solution.

![Average percentage of best LP-solution found](image)

Figure 14: The difference in performance between IPCS-Master solution and the Best LNS Solution found at each IPCS-Master iteration, i.e. at every update of the dual solution.

## 5.2 Branch-and-Price

GCG is the software that we intended to use for benchmark. GCG solves MIP problems with Branch-and-price method. We provided GCG interactive shell with an instance file of the compact formulation (14) and a decomposition file. The decomposition was made as described in Chapter 3.1.

GCG found the instances challenging to solve, consequently we let the computational experiment run for 24 hours before terminating. Table 4 lists the results of the computational experiment evaluating the compact formulation of $P|r_j, \text{deadline}, M_j, s_{jk}, \sum \omega_j U_j$ using GCG and its Branch-and-Price method. The
The **Gap** column in Table 4 represents the relative gap between the best integer solution found and the LP-relaxation of the compact formulation of the problem, i.e. an LP-relaxation of all integer variables. The instances of size 100 were excluded because of how challenging the problems were to solve.

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>nr.</th>
<th>nr. machines</th>
<th>t[s]</th>
<th>IP-sol</th>
<th>Gap [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>3</td>
<td>11</td>
<td>3</td>
<td>86 400</td>
<td>1087</td>
<td>108.65</td>
</tr>
<tr>
<td>70</td>
<td>5</td>
<td>20</td>
<td>3</td>
<td>86 400</td>
<td>865</td>
<td>173.87</td>
</tr>
<tr>
<td>80</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>86 400</td>
<td>759</td>
<td>201.05</td>
</tr>
<tr>
<td>80</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>86 400</td>
<td>1186</td>
<td>129.85</td>
</tr>
<tr>
<td>80</td>
<td>5</td>
<td>10</td>
<td>3</td>
<td>86 400</td>
<td>1291</td>
<td>99.46</td>
</tr>
<tr>
<td>90</td>
<td>3</td>
<td>25</td>
<td>3</td>
<td>86 400</td>
<td>24</td>
<td>10583.33</td>
</tr>
<tr>
<td>90</td>
<td>4</td>
<td>23</td>
<td>3</td>
<td>86 400</td>
<td>1130</td>
<td>157.61</td>
</tr>
<tr>
<td>90</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>86 400</td>
<td>967</td>
<td>183.76</td>
</tr>
</tbody>
</table>

Table 4: Results of Branch-and-price method

Table 5 lists the solutions of the IPCS method and the runtime of the computational experiment. We use this data to compare with the results of the GCG software.

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>nr.</th>
<th>nr. machines</th>
<th>t[s]</th>
<th>IP-sol</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>3</td>
<td>11</td>
<td>3</td>
<td>2 148</td>
<td>2034</td>
</tr>
<tr>
<td>70</td>
<td>5</td>
<td>20</td>
<td>3</td>
<td>3 001</td>
<td>2305</td>
</tr>
<tr>
<td>80</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3 177</td>
<td>2158</td>
</tr>
<tr>
<td>80</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>2 826</td>
<td>2408</td>
</tr>
<tr>
<td>80</td>
<td>5</td>
<td>10</td>
<td>3</td>
<td>8 288</td>
<td>2353</td>
</tr>
<tr>
<td>90</td>
<td>3</td>
<td>25</td>
<td>3</td>
<td>9 729</td>
<td>2393</td>
</tr>
<tr>
<td>90</td>
<td>4</td>
<td>23</td>
<td>3</td>
<td>7 796</td>
<td>2296</td>
</tr>
<tr>
<td>90</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>12 092</td>
<td>2467</td>
</tr>
<tr>
<td>90</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>11 923</td>
<td>2331</td>
</tr>
</tbody>
</table>

Table 5: Results of IPCS method

Figure 15 depict the differences in performance between the IPCS method and GCG. The color and style of the markers separate the methods and the size separate the number of secondary resources. The horizontal axis represents instance size and the vertical axis represent the value of the integer solution. The figure shows that GCG gives poor solutions and very varying results while IPCS-Master reaches higher values of the integer solution. Note the significant shorter runtimes of the IPCS method compared to GCG.
Figure 15: Instances evaluated on GCG and IPCS with 3 machines available, provided with the runtimes of the computational experiment for the different evaluated methods listed in Table 4 and 5.
6. Discussion

The discussion is based on the implementation of IPCS made for this thesis. Conclusions drawn from the results and computational experiments are presented in Chapter 6.1 and the suggestions for future improvements of the design are presented in Chapter 6.2.

6.1 Conclusion

The aim of this thesis work was to implement the method IPCS that combines the use of a Mixed Integer Programming solver with a heuristic search approach. An important part of the work was to investigate how to efficiently implement this method and to choose search strategies and make parameter settings. This chapter presents the conclusion drawn from the results and computational experiments made with IPCS and the GCG software.

6.1.1 IPCS

We have shown that the design of the implementation of IPCS can be applied to solve $P|r_j, \text{deadline}, M_j, s_j|\sum \omega_j U_j$ problems. The result shows that IPCS efficiently finds feasible solutions of high quality.

Based on the implemented design of the IPCS method there are three main arguments for using IPCS-Master to update the dual variable values:

- It generally takes fewer pricing problem iterations for reaching a near-optimal integer solution.
- Columns included in the near-optimal integer solution can be generated using different values of $\tilde{u}$.
- GIP-CG can be faster to solve than the pricing problem in the standard linear programming column generation.

The computational experiments performed during the development IPCS indicate that the introduction of IPCS-Master generally leads to fewer pricing problem iterations to reach a near-optimal integer solution. Furthermore, the experiments indicate that to update the dual variable values used in GIP-CG and to decrease the value of $\gamma$ are substantial for the generation of new columns that benefits an integer solution rather than an LP-solution. See Chapter 4.2 for details of the development of IPCS.

Smaller values of $\gamma$ takes the objective function values of GIP-CG closer to the original cost. Letting the penalty parameter $M$ take large values in this situation influence the search of a new column such that it includes the most profitable jobs that are not already in the Current Solution. The computational experiments
indicate that this type of collaboration between $\gamma$ and $M$ prevent regeneration of columns.

The introduction of IPCS-Master to update the dual variable values is a large part of the prosperity of IPCS. However, the need of solving IPCS-Master to get an IP solution may not be of such importance. As one can see in Figure 14, LNS Best Solution gives almost the same objective value as IPCS-Master.

The computational experiments evaluating IPCS as implemented in this thesis gives promising results and hopefully they inspire to further research.

6.1.2 GCG

The statistical output files indicate that GCG do not find an optimal LP-solution to the master problem (7). However, it does find a weaker optimistic bound which is based on the optimistic bound from the LP-relaxation of SUB. This bound on (7) is poor since the strength of the bound obtained from the LP-relaxation of a master problem relies on the integrality of the variables in SUB. In this case, when only the LP-relaxation of SUB can be solved to optimality, the resulting bound on (7) has the same type of strength as the LP-relaxation of the compact formulation.

As one can see in Table 4 the preliminary experiments with GCG do not provide near-optimal solutions. This indicates how challenging it is to find optimal solutions and prove their optimality. Since the preliminary experiments do not even find a solution to the LP-relaxation of the master problem we do not use the results for benchmark.

6.2 Future Work

The design of the implementation of IPCS has many opportunities. There are some straightforward parts to implement and then there are those more subtle parts that can be adjusted to reach wanted accuracy. Below there are some of the thoughts that may be of interest in future research since they have not been explored in this work.

Stabilization

The dual solutions are, in the first iterations, oscillating and giving poor information to the pricing problem, this can be observed in the first column generations in Figure 11a. This behavior of the dual solution can be stabilized by putting an upper and a lower bound on the values and penalize if the solution appears outside these boundaries [11]. A stabilization method may be of interest to investigate since it can offer a more efficient search for columns.
Implementation design

A deeper understanding of the dynamic parameter values could be obtained by looking at what parameter values that regenerate columns included in the IPCS-Master solution. If it was possible to get knowledge of which columns are associated to which dynamic parameter values, then it would be possible to know if there are certain parameter values that produce columns that are more frequently a part of the IPCS-Master solution.

For further implementation designs, it would be of interest to see how the termination of the pricing problems affect the prosperity of the method reaching such good integer solutions. Because of the difficulty of the problem, it has only been superficially investigated how the termination affects the result. It may be of interest in future work to see what the termination actually change during the search for columns.

There are many efficient ways of initializing LP-CG such as big M methods or by adding artificial variables. This is made in many of the referenced works. Better initialization leads to a quicker start for both the LP-CG method and the LNS-CG method. This thesis work did not investigate this since the main focus was to derive a strategy for the dynamic parameter values of IPCS. Initializing with one column derived by solving the original single machine problem was quick to implement.

Benchmark

GCG is a software with a lot of potential, the software does not have such an easy to use API as Gurobi Optimizer but gives flexible implementation possibilities. This thesis work did not include enough time to fully explore the potential of using GCG. However, there are three cornerstones that we would have explored if time had allowed: First, the compact model can preferably be derived without symmetries. If that is not done it would be of interest to get GCG to detect and aggregate these symmetries. Secondly, since the pricing problem (single machine problem with modified costs) is hard to solve, a plug-in method for solving this problem faster might be of interest. Lastly, it may be of interest to work on the branching rules and strategies for this specific problem.
References


Copyright

The publishers will keep this document online on the Internet - or its possible replacement – for a period of 25 years from the date of publication barring exceptional circumstances. The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for their own use and to use it unchanged for any noncommercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtidiga ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämlig som upphovsmann i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

© 2019, Julia Jobson
# Appendices

## A. Result tables

<table>
<thead>
<tr>
<th>Method</th>
<th>Size</th>
<th>Percentage</th>
<th>Jobs</th>
<th>Machines</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPCS</td>
<td>60</td>
<td>99.514892</td>
<td>45.0</td>
<td>2</td>
<td>m0060,n01,011</td>
</tr>
<tr>
<td>IPCS</td>
<td>60</td>
<td>98.910966</td>
<td>46.0</td>
<td>2</td>
<td>m0060,n04,024</td>
</tr>
<tr>
<td>IPCS</td>
<td>60</td>
<td>98.422374</td>
<td>44.0</td>
<td>2</td>
<td>m0060,n05,009</td>
</tr>
<tr>
<td>IPCS</td>
<td>70</td>
<td>100.00000</td>
<td>59.0</td>
<td>3</td>
<td>m0070,n01,011</td>
</tr>
<tr>
<td>IPCS</td>
<td>70</td>
<td>99.54199</td>
<td>47.0</td>
<td>2</td>
<td>m0070,n03,023</td>
</tr>
<tr>
<td>IPCS</td>
<td>70</td>
<td>99.568035</td>
<td>66.0</td>
<td>3</td>
<td>m0070,n04,005</td>
</tr>
<tr>
<td>IPCS</td>
<td>70</td>
<td>99.138187</td>
<td>47.0</td>
<td>2</td>
<td>m0070,n04,010</td>
</tr>
<tr>
<td>IPCS</td>
<td>70</td>
<td>99.725394</td>
<td>63.0</td>
<td>3</td>
<td>m0070,n06,020</td>
</tr>
<tr>
<td>IPCS</td>
<td>80</td>
<td>98.889441</td>
<td>67.0</td>
<td>3</td>
<td>m0080,n03,001</td>
</tr>
<tr>
<td>IPCS</td>
<td>80</td>
<td>98.615146</td>
<td>47.0</td>
<td>2</td>
<td>m0080,n01,010</td>
</tr>
<tr>
<td>IPCS</td>
<td>80</td>
<td>97.720570</td>
<td>46.0</td>
<td>2</td>
<td>m0080,n05,013</td>
</tr>
<tr>
<td>IPCS</td>
<td>80</td>
<td>99.725394</td>
<td>63.0</td>
<td>3</td>
<td>m0080,n06,020</td>
</tr>
<tr>
<td>IPCS</td>
<td>90</td>
<td>98.889441</td>
<td>67.0</td>
<td>3</td>
<td>m0090,n03,011</td>
</tr>
<tr>
<td>IPCS</td>
<td>90</td>
<td>98.327778</td>
<td>45.0</td>
<td>2</td>
<td>m0090,n01,023</td>
</tr>
<tr>
<td>IPCS</td>
<td>90</td>
<td>99.280700</td>
<td>68.0</td>
<td>3</td>
<td>m0090,n04,025</td>
</tr>
<tr>
<td>IPCS</td>
<td>90</td>
<td>96.171152</td>
<td>47.0</td>
<td>2</td>
<td>m0090,n04,006</td>
</tr>
<tr>
<td>IPCS</td>
<td>90</td>
<td>98.959478</td>
<td>68.0</td>
<td>3</td>
<td>m0090,n05,023</td>
</tr>
<tr>
<td>IPCS</td>
<td>90</td>
<td>98.298830</td>
<td>66.0</td>
<td>3</td>
<td>m0090,n05,007</td>
</tr>
<tr>
<td>IPCS</td>
<td>90</td>
<td>97.151227</td>
<td>48.0</td>
<td>2</td>
<td>m0090,n05,019</td>
</tr>
<tr>
<td>IPCS</td>
<td>100</td>
<td>98.520545</td>
<td>70.0</td>
<td>3</td>
<td>m100,n03,017</td>
</tr>
<tr>
<td>IPCS</td>
<td>100</td>
<td>98.091117</td>
<td>47.0</td>
<td>2</td>
<td>m100,n03,020</td>
</tr>
<tr>
<td>IPCS</td>
<td>100</td>
<td>95.886757</td>
<td>45.0</td>
<td>2</td>
<td>m100,n05,014</td>
</tr>
<tr>
<td>IPCS</td>
<td>100</td>
<td>99.016322</td>
<td>72.0</td>
<td>3</td>
<td>m100,n04,020</td>
</tr>
<tr>
<td>IPCS</td>
<td>100</td>
<td>97.857143</td>
<td>44.0</td>
<td>2</td>
<td>m100,n05,015</td>
</tr>
<tr>
<td>IPCS</td>
<td>100</td>
<td>98.172684</td>
<td>68.0</td>
<td>3</td>
<td>m100,n05,018</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>60</td>
<td>66.478498</td>
<td>25.0</td>
<td>2</td>
<td>m0060,n03,011</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>60</td>
<td>66.601676</td>
<td>25.0</td>
<td>2</td>
<td>m0060,n04,024</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>60</td>
<td>61.150950</td>
<td>24.0</td>
<td>2</td>
<td>m0060,n05,009</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>70</td>
<td>64.95019</td>
<td>37.0</td>
<td>3</td>
<td>m0070,n03,011</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>70</td>
<td>57.312247</td>
<td>47.0</td>
<td>2</td>
<td>m0070,n05,023</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>70</td>
<td>70.539557</td>
<td>43.0</td>
<td>3</td>
<td>m0070,n04,005</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>70</td>
<td>56.688742</td>
<td>24.0</td>
<td>2</td>
<td>m0070,n05,009</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>70</td>
<td>92.052064</td>
<td>47.0</td>
<td>2</td>
<td>m0070,n05,013</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>70</td>
<td>69.31594</td>
<td>43.0</td>
<td>3</td>
<td>m0070,n05,020</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>80</td>
<td>62.938307</td>
<td>46.0</td>
<td>3</td>
<td>m0070,n05,005</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>80</td>
<td>58.612433</td>
<td>24.0</td>
<td>2</td>
<td>m0080,n03,010</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>80</td>
<td>67.930349</td>
<td>43.0</td>
<td>2</td>
<td>m0080,n05,006</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>80</td>
<td>59.281123</td>
<td>24.0</td>
<td>2</td>
<td>m0080,n04,022</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>80</td>
<td>64.134397</td>
<td>46.0</td>
<td>3</td>
<td>m0080,n05,010</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>80</td>
<td>67.931858</td>
<td>24.0</td>
<td>2</td>
<td>m0080,n05,015</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>90</td>
<td>56.957506</td>
<td>45.0</td>
<td>2</td>
<td>m0090,n03,023</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>90</td>
<td>68.493305</td>
<td>47.0</td>
<td>3</td>
<td>m0090,n05,025</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>90</td>
<td>58.680218</td>
<td>24.0</td>
<td>2</td>
<td>m0090,n05,006</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>90</td>
<td>69.783116</td>
<td>45.0</td>
<td>3</td>
<td>m0090,n04,023</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>90</td>
<td>67.345744</td>
<td>46.0</td>
<td>3</td>
<td>m0090,n05,007</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>90</td>
<td>60.943344</td>
<td>23.0</td>
<td>2</td>
<td>m0090,n05,019</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>100</td>
<td>70.531098</td>
<td>49.0</td>
<td>3</td>
<td>m100,n03,017</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>100</td>
<td>55.27702</td>
<td>24.0</td>
<td>2</td>
<td>m100,n05,020</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>100</td>
<td>54.635202</td>
<td>24.0</td>
<td>2</td>
<td>m100,n04,014</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>100</td>
<td>65.282557</td>
<td>48.0</td>
<td>3</td>
<td>m100,n04,020</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>100</td>
<td>55.204082</td>
<td>21.0</td>
<td>2</td>
<td>m100,n05,015</td>
</tr>
<tr>
<td>LP-CG and IP</td>
<td>100</td>
<td>87.273871</td>
<td>46.0</td>
<td>3</td>
<td>m100,n05,018</td>
</tr>
</tbody>
</table>

Table 6: The table list results from the LP-CG and IP method and the IPCS method which is used in the distribution diagrams in Chapter 5. The Percentage column represent the quality of solutions after approximately 477 pricing problem iterations. Jobs are the number of jobs scheduled in the integer solution and size represent the instance size.